**write a program to calculate Fibonacci numbers and analyse their time and space complexity.**

#include<stdio.h>

**int** fib(**int** n)

{

  /\* Declare an array to store Fibonacci numbers. \*/

**int** f[n+2];   // 1 extra to handle case, n = 0

**int** i;

  /\* 0th and 1st number of the series are 0 and 1\*/

  f[0] = 0;

  f[1] = 1;

**for** (i = 2; i <= n; i++)

  {

      /\* Add the previous 2 numbers in the series

         and store it \*/

      f[i] = f[i-1] + f[i-2];

  }

**return** f[n];

}

**int** main ()

{

**int** n = 9;

**printf**("%d", fib(n));

**getchar**();

**return** 0;

}

**2. Write a program to sort jobs based an maximum profits with**

**deadlines using greedy method.**

// C program for the above approach

#include <stdbool.h>

#include <stdio.h>

#include <stdlib.h>

// A structure to represent a job

typedef struct Job {

char id; // Job Id

int dead; // Deadline of job

int profit; // Profit if job is over before or on

// deadline

} Job;

// This function is used for sorting all jobs according to

// profit

int compare(const void\* a, const void\* b)

{

Job\* temp1 = (Job\*)a;

Job\* temp2 = (Job\*)b;

return (temp2->profit - temp1->profit);

}

// Find minimum between two numbers.

int min(int num1, int num2)

{

return (num1 > num2) ? num2 : num1;

}

// Returns maximum profit from jobs

void printJobScheduling(Job arr[], int n)

{

// Sort all jobs according to decreasing order of profit

qsort(arr, n, sizeof(Job), compare);

int result[n]; // To store result (Sequence of jobs)

bool slot[n]; // To keep track of free time slots

// Initialize all slots to be free

for (int i = 0; i < n; i++)

slot[i] = false;

// Iterate through all given jobs

for (int i = 0; i < n; i++) {

// Find a free slot for this job (Note that we start

// from the last possible slot)

for (int j = min(n, arr[i].dead) - 1; j >= 0; j--) {

// Free slot found

if (slot[j] == false) {

result[j] = i; // Add this job to result

slot[j] = true; // Make this slot occupied

break;

}

}

}

// Print the result

for (int i = 0; i < n; i++)

if (slot[i])

printf("%c ", arr[result[i]].id);

}

// Driver's code

int main()

{

Job arr[] = { { 'a', 2, 100 },

{ 'b', 1, 19 },

{ 'c', 2, 27 },

{ 'd', 1, 25 },

{ 'e', 3, 15 } };

int n = sizeof(arr) / sizeof(arr[0]);

printf(

"Following is maximum profit sequence of jobs \n");

// Function call

printJobScheduling(arr, n);

return 0;

}

**3. Write a program to solve fractional knapscak using greedy method.**

// Java program to solve fractional Knapsack Problem

import java.io.\*;

import java.util.Arrays;

import java.util.Comparator;

// Greedy approach

class FractionalKnapSack {

// Function to get maximum value

private static double getMaxValue(ItemValue[] arr,

int capacity)

{

// Sorting items by value/weight ratio;

Arrays.sort(arr, new Comparator<ItemValue>() {

@Override

public int compare(ItemValue item1,

ItemValue item2)

{

double cpr1

= new Double((double)item1.value

/ (double)item1.weight);

double cpr2

= new Double((double)item2.value

/ (double)item2.weight);

if (cpr1 < cpr2)

return 1;

else

return -1;

}

});

double totalValue = 0d;

for (ItemValue i : arr) {

int curWt = (int)i.weight;

int curVal = (int)i.value;

if (capacity - curWt >= 0) {

// this weight can be picked while

capacity = capacity - curWt;

totalValue += curVal;

}

else {

// Item cant be picked whole

double fraction

= ((double)capacity / (double)curWt);

totalValue += (curVal \* fraction);

capacity

= (int)(capacity - (curWt \* fraction));

break;

}

}

return totalValue;

}

// Item value class

static class ItemValue {

int value, weight;

// Item value function

public ItemValue(int val, int wt)

{

this.weight = wt;

this.value = val;

}

}

// Driver code

public static void main(String[] args)

{

ItemValue[] arr = { new ItemValue(60, 10),

new ItemValue(100, 20),

new ItemValue(120, 30) };

int capacity = 50;

double maxValue = getMaxValue(arr, capacity);

// Function call

System.out.println(maxValue);

}

}

**4 Write a program to generate Binomial coefficient using dynamic programming.**

// A Dynamic Programming based solution

// that uses table C[][] to

// calculate the Binomial Coefficient

#include <stdio.h>

// Prototype of a utility function that

// returns minimum of two integers

int min(int a, int b);

// Returns value of Binomial Coefficient C(n, k)

int binomialCoeff(int n, int k)

{

int C[n + 1][k + 1];

int i, j;

// Calculate value of Binomial Coefficient

// in bottom up manner

for (i = 0; i <= n; i++) {

for (j = 0; j <= min(i, k); j++) {

// Base Cases

if (j == 0 || j == i)

C[i][j] = 1;

// Calculate value using

// previously stored values

else

C[i][j] = C[i - 1][j - 1] + C[i - 1][j];

}

}

return C[n][k];

}

// A utility function to return

// minimum of two integers

int min(int a, int b) { return (a < b) ? a : b; }

/\* Drier program to test above function\*/

int main()

{

int n = 5, k = 2;

printf("Value of C(%d, %d) is %d ", n, k,

binomialCoeff(n, k));

return 0;

}